Run Time Support Libraries

REAL Math Facility	Emulator or Actual Chip Used	Link-List Should Include the Specifications Below (Not Necessarily In the Order Shown) After Object Modules
None	neither	none
All Floating-Point in PL/M-86 *only*	emulator	E8087.LIB, PE8087
With Some Modules That Use Floating-Point *not* in PL/M-86	emulator	E8087.LIB, E8087
Any	actual 8087 chip	8087.LIB

Hex-ASCII Table

NUL	00	+	2B	V	56
SOH	01	,	2C	W	57
STX	02	-	2D	X	58
ETX	03	.	2E	Y	59
EOT	04	/	2F	Z	5A
ENQ	05	0	30	[	5B
ACK	06	1	31	\	5C
BEL	07	2	32	]	5D
BS	08	3	33	^ (↑)	5E
HT	09	4	34	_ (←)	5F
LF	0A	5	35	`	60
VT	0B	6	36	a	61
FF	0C	7	37	b	62
CR	0D	8	38	c	63
SO	0E	9	39	d	64
SI	0F	:	3A	e	65
DLE	10	;	3B	f	66
DC1 (X-ON)	11	<	3C	g	67
DC2 (TAPE)	12	=	3D	h	68
DC3 (X-OFF)	13	>	3E	i	69
DC4 (TAPE)	14	?	3F	j	6A
NAK	15	@	40	k	6B
SYN	16	A	41	l	6C
ETB	17	B	42	m	6D
CAN	18	C	43	n	6E
EM	19	D	44	o	6F
SUB	1A	E	45	p	70
ESC	1B	F	46	q	71
FS	1C	G	47	r	72
GS	1D	H	48	s	73
RS	1E	I	49	t	74
US	1F	J	4A	u	75
SP	20	K	4B	v	76
!	21	L	4C	w	77
"	22	M	4D	x	78
#	23	N	4E	y	79
$	24	O	4F	z	7A
%	25	P	50	{	7B
&	26	Q	51	|	7C
'	27	R	52	} (ALT MODE)	7D
(	28	S	53	~	7E
)	29	T	54	DEL (RUB OUT)	7F
*	2A	U	55		

3065 Bowers Avenue, Santa Clara, California 95051
(408) 987-8080
Printed in U.S.A.
3210/600/0887/K2/NR/DTO
SOFTWARE
122412-002

intel *Development Solutions*

PL/M-86 Pocket Reference for DOS Systems

Order Number: 122412-002

CONTENTS

	PAGE
Notational Conventions	1
Compiler Invocation	2
Lexical Elements	2
Modules and the Main Program	3
Declarations	3
Statements and Related Constructs	5
Expressions	7
Built-In Procedures and Functions	9
iAPX 86/88 Built-In Variables	11
iAPX 86/88 Built-In Functions	12
8087 Built-In Procedures and Functions	12
Compiler Controls	12
Run Time Support Libraries	14
Hex-ASCII Table	14

All Mnemonics Copyright © Intel Corp., 1985
All rights reserved.

Notational Conventions

THIS TYPE	Use these keywords, letters, symbols and punctuation verbatim. Upper or lower case is acceptable.
italics	Substitute language elements or constructs for true terms.
[ ]	Optional constructs.
[ ]...	Optional constructs that can be repeated any number of times.
{ }	Alternate constructs. Choose any one of the constructs enclosed in the braces.
/* text enclosed */	Text enclosed is a prose definition of the construct.

When two adjacent items must be concatenated, they appear with no space between them. A blank space between two items indicates that the two items may be separated by one or more blanks.

Compiler Invocation

The general form of the command line used to invoke the compiler is:

[*directory*] plm86 *source* [*controls*] <cr>

where

directory	is the portion of the pathname that identifies the device and directories that contain the file plm86.
plm286	is the name of the PL/M-86 compiler. Note that it is not necessary to specify the .EXE extension.
source	is the name of the file that contains the source.
controls	are the optional compiler controls.
<cr>	is a carriage return.

Lexical Elements

token (*delimiter* | *identifier* | *reserved-word* | *numeric-constant* | *string*)

delimiter /*any one of the following:
+ − * / < > = : ; . , ') (
@ <> <= >= := */

identifier *letter* [(*letter* | *decimal-digit* | $ | _)] ...

numeric-constant (*binary-number* | *octal-number* | *decimal-number* | *hexadecimal-number* | *floating-point-number*)

binary-number *binary-digit* [(*binary-digit* | $)] ... B

octal number *octal-digit* [(*octal-digit* | $)] ... (O | Q)

decimal-number *decimal-digit* [(*decimal-digit* | $)] ...[D]

hexadecimal-number *decimal-digit* [(*hexadecimal-digit* | $)] ... H

floating-point number *digit-string* *fractional-part* [*exponent-part*]

fractional part .[*digit-string*]

exponent part E [(+ | −)] *digit-string*

digit-string *decimal-digit* [(*decimal-digit* | $)] ...

string 'string-body-element'

string-body-element (*non-quote-character* | '')

plm-text [(*token* | *separator*)] ...

separator (*blank* | *comment*)

comment /*[*character*] ... */

Modules and the Main Program

compilation *module* | EOF |

module *module-name* : *simple-do-block*

module-name *identifier*

Declarations

declaration (*declare-statement* | *procedure-definition*)

declare-statement DECLARE (*factored-element* | *unfactored-element*) ...

unfactored-element (*variable-element* | *literal-element* | *label-element*)

factored-element (*factored-variable-element* | *factored-label-element*)

variable-element { *non-based-name* | *based-name* BASED *base-specifier* *variable-type* [*variable-attributes*] } [*array-specifier*]

non-based name *variable-name-identifier*

based-name *variable-name-identifier*

base-specifier *identifier* [, *identifier*]

variable-attribute { [PUBLIC] [*locator*] [*initialization*] | [EXTERNAL] [*constant-attribute*] }

locator AT (*expression*)

constant-attribute DATA

array-specifier { (*numeric-constant*) | (*) }

variable-type { *basic-type* | *structure type* }

basic-type { INTEGER | REAL | POINTER | SELECTOR | BYTE | WORD | DWORD }

label-element *identifier* LABEL [{ PUBLIC | EXTERNAL }]

literal-elements *identifier* LITERALLY *string*

factored-variable-element (*variable-name-specifier* [, *variable-name-specifier*] ...) [*explicit-dimensions*] *variable-type* [*variable-attribute*]

factored-label-element (*identifier* [, *identifier*]...) LABEL [{ PUBLIC | EXTERNAL }]

structure-type STRUCTURE (*member-element* [, *member-element*]...)

member-element *member-name* [*explicit-dimension*] *basic-type*

member-name *identifier*

procedure-definition *procedure-statment* [*declaration* ...] [*unit*...] *ending*

procedure-statement *procedure-name* : PROCEDURE [*formal-parameter-list*] [*procedure-type*] [*procedure-attribute*];

procedure-name *identifier*

procedure-type *basic-type*

formal-parameter-list (*formal-parameter* [, *formal-parameter*] ...)

formal-parameter *identifier*

procedure-attributes { { *interrupt* | EXTERNAL } ... | { *interrupt* | PUBLIC | REENTRANT } ... }

interrupt INTERRUPT

initialization { INITIAL | DATA } (*initial-value* [, *initial-value*] ...)

initial-value { *expression* | *string* }

Statements and Related Constructs

unit { *conditional-clause* | *do-block* | *basic-statement* | *label-definition unit* }

basic-statement { *assignment-statement* | *call-statement* | *goto-statement* | *null-statement* | *return-statement* | ***8086-dependent-statement*** }

scoping-statement { *simple-do-statement* | *do-case-statement* | *do-while-statement* | *iterative-do-statement* | *end-statement* | *procedure-statement* }

label-definition *identifier* :

assignment-statement *left-part* = *expression* ;

left-part *variable-reference* [, *variable-reference*]...

call-statement CALL *simple-variable* [*parameter-list*] ;

parameter-list (*expression* [, *expression*]...)

simple-variable { *identifier* | *identifier* . *identifier* }

goto-statement { GOTO | GO TO } *identifier* ;

null-statement ;

return-statement { *typed-return* | *untyped-return* }

typed-return RETURN *expression* ;

untyped-return RETURN;

8086-dependent-statement { *disable-statement* | *enable-statement* | *halt-statement* | *cause-interrupt-statement* }

disable-statement DISABLE;

enable-statement ENABLE;

halt-statement HALT;

cause-interrupt-statement CAUSE$INTERRUPT (*numeric-constant*);

simple-do-statement DO;

do-case-statement DO CASE *expression* ;

do-while-statement DO WHILE *expression* ;

iterative-do-statement DO *index-part to-part* [*by-part*];

index-part *index-variable* = *start-expression*

to-part TO *bound-expression*

by-part BY *step-expression*

index-variable *simple-variable*

start-expression *expression*

bound-expression *expression*

step-expression *expression*

end-statement END [*identifier*];

procedure-statement *procedure name* : PROCEDURE [*formal-parameter-list*] [*procedure-type*] [*procedure attributes*];

conditional-clause { *if-condition true-unit* | *if-condition true-element* ELSE *false-element* }

if condition IF *expression* THEN

true-element { [*label-definition*]... *do-block* | [*label-definition*]... *basic-statement* }

false-element *unit*

true-unit *unit*

do-block { *simple-do-block* | *do-case-block* | *do-while-block* | *iterative-do-block* }

simple-do-block *simple-do-statement* [*declaration*]... [*unit*]... *ending*

ending [*label definition*]... *end statement*

do-case-block *do-case-statement* [*unit*]... *ending*

do-while-block *do-while-statement* [*unit*]... *ending*

iterative-do-block *iterative-do-statement* [*unit*]... *ending*

Expressions

primary { *constant* | *variable-reference* | *location-reference* | *subexpression* }

subexpression (*expression*)

constant { *numeric-constant* | *string* }

variable-references { *data-reference* | *function-reference* }

data-reference *name* [*subscript*] [*member-specifier*]

subscript (*expression*)

member-specifier .*member-name* [*subscript*]

function-reference *name* [*actual-parameters*]

actual-parameters (*expression* [, *expression*]...)

member-name *identifier*

name *identifier*

location-reference { { @ | . } *constant-list* | { @ | . } *variable-reference* }

constant-list (*constant* [, *constant*]...)

operator { *logical-operator* | *relational-operator* | *arithmetic-operator* }

logical-operator { AND | OR | NOT | XOR }

relational-operator /* any one of the following:

< > <= >= <> = */

arithmetic-operator /* any one of the following:

+ − PLUS MINUS * / MOD */

expression { *logical-expression* | *embedded-assignment* }

embedded-assignment *variable-reference* := *logical-expression*

logical-expression { *logical-factor* | *logical-expression or-operator logical-factor* }

or-operator { OR | XOR }

logical-factor { *logical-secondary* | *logical-factor and-operator logical-secondary* }

and-operator AND

logical-secondary [*not-operator*] *logical-primary*

not-operator NOT

logical-primary *arithmetic-expression* [*relational-operator* *arithmetic-expression*]

arithmetic-expression { *term* | *arithmetic-expression adding-operator term* }

adding-operator { + | − | PLUS | MINUS }

term { *secondary* | *term multiplying-operator secondary* }

multiplying-operator { * | / | MOD }

secondary { *unary-minus* | *unary-plus* } *primary*

unary-minus −

unary-plus +

Built-In Procedures and Functions

Identifier	Parameter List*	Returned Value	Operation
ABS	(*real-expr*)	REAL	Returns absolute value of REAL expression
BUILD$PTR	(*base,offset*)	POINTER	Returns POINTER constructed with base SELECTOR and WORD offset
CMPB	(*source1,source2,count*)	WORD	Compares two BYTE strings to find first element that does not match
CMPW	(*source1,source2,count*)	WORD	Compares two WORD strings to find first element that does not match
DOUBLE	(*byte-expr*) (*word-expr*) (*dword-expr*)	WORD DWORD DWORD	Converts BYTE value to WORD value; WORD value to DWORD value; DWORD value unchanged
FINDB	(*source,target,count*)	WORD	Searches a source BYTE string for first element that matches value of target
FINDRB	(*source,target,count*)	WORD	Searches source BYTE string for last element that matches value of target
FINDRW	(*source,target,count*)	WORD	Searches source WORD string for last element that matches value of target
FINDW	(*source,target,count*)	WORD	Searches a source WORD string for first element that matches value of target
FIX	(*real-expr*)	INTEGER	Converts REAL value to INTEGER value (rounds toward zero) within range ± 32767
FLOAT	(*int-expr*)	REAL	Converts INTEGER value to REAL value
HIGH	(*byte-expr*) (*word-expr*) (*dword-expr*)	BYTE BYTE WORD	Converts BYTE value to zero; WORD value to high-order BYTE of WORD; DWORD value to high-order WORD of DWORD
IABS	(*int-expr*)	INTEGER	Returns absolute value of INTEGER expression
INPUT	(*port-number*)	BYTE	Returns BYTE value in specified input port
INT	(*word-expr*)	INTEGER	Converts WORD value to INTEGER; bit pattern is unchanged, but interpreted as positive
INTERRUPT$PTR	(*interrupt-proc-name*)	POINTER	Returns interrupt entry point of specified interrupt procedure

Built-In Procedures and Functions (Cont'd.)

Identifier	Parameter List*	Returned Value	Operation
INWORD	(*port-number*)	WORD	Returns WORD value in specified input port
LAST	(*array-variable-ref*)	WORD	Returns subscript of last element in array
LENGTH	(*array-variable-ref*)	WORD	Returns the number of elements in array
LOCKSET	(*lockptr,new-value*)	BYTE	Implements a simple software lock; returns old value of BYTE pointed to by lockptr
LOW	(*byte-expr*) (*word-expr*) (*dword-expr*)	BYTE BYTE WORD	BYTE value unchanged; converts WORD value to low-order BYTE of WORD; converts DWORD value to low-order WORD of DWORD
MOVB	(*source,dest,count*)	none	Copies a BYTE string from one location to another
MOVE	(*count,source,dest*)	none	Copies a BYTE string from one location to another; provides compatibility with PL/M-80
MOVRB	(*source,dest,count*)	none	Copies a BYTE string from one location to another in descending order
MOVRW	(*source,dest,count*)	none	Copies a WORD string from one location to another in descending order
MOVW	(*source,dest,count*)	none	Copies a WORD string from one location to another
NIL	none	POINTER	Returns POINTER value with all bytes set to zero
OFFSET$OF	(*pointer*)	WORD	Returns offset portion of specified POINTER
ROL	(*pattern,count*)	BYTE, WORD or DWORD	Rotates an 8-, 16-, or 32-bit pattern to the left
SAL	(*pattern,count*)	INTEGER	Shifts the 16-bit pattern to the left with zero fill
SAR	(*pattern,count*)	INTEGER	Shifts the 16-bit pattern to the right with sign bit fill
SELECTOR$OF	(*pointer*)	SELECTOR	Returns base portion of specified POINTER
SETB	(*new-value,dest,count*)	none	Sets each element of a BYTE string to a single specified value
SETW	(*new-value,dest,count*)	none	Sets each element of a WORD string to a single specified value
SET$INTERRUPT	(*constant,interrupt-proc-name*)	none	Sets the specified interrupt vector to point to the specified interrupt procedure
SHL	(*pattern,count*)	BYTE, WORD or DWORD	Shifts an 8-, 16-, or 32-bit pattern to the left with zero fill

Built-In Procedures and Functions (Cont'd.)

Identifier	Parameter List*	Returned Value	Operation
SHR	(*pattern,count*)	BYTE, WORD or DWORD	Shifts an 8-, 16-, or 32-bit pattern to the right with zero fill
SIGNED	(*work-expr*)	INTEGER	Converts WORD value to INTEGER; bit pattern is unchanged
SIZE	(*variable-ref*)	WORD	Returns the number of bytes occupied by the variable
SKIPB	(*source,target,count*)	WORD	Searches source BYTE string for first element that does not matach value of target
SKIPRB	(*source,target,count*)	WORD	Searches source BYTE string for last element that does not match value of target
SKIPRW	(*source,target,count*)	WORD	Searches source WORD string for last element that does not match value of target
SKIPW	(*source,target,count*)	WORD	Searches source WORD string for first element that does not match value of target
TIME	(*word-expr*)	none	Causes a time delay of specified duration
UNSIGN	(*int-expr*)	WORD	Converts INTEGER value to WORD; bit pattern is unchanged
XLAT	(*source,dest,count,table*)	none	Translates source BYTE string using translation table to produce destination BYTE string

*For readability, optional blanks have been omitted from the syntax.

iAPX 86/88 Built-In Variables

Identifier	Meaning
FLAGS	A WORD variable that provides access to the iAPX 86 hardware flags register
MEMORY	A BYTE array of unspecified length that represents an uninitialized segment of the iAPX 86/88 storage
OUTPUT	A BYTE array of 65536 elements that represent the hardware output ports of the iAPX 86/88; can only be used on the left-hand side of an assignment statement
OUTWORD	A WORD array of 65536 elements that represent the hardware output ports of the iAPX 86/88; can only be used on the left-hand side of an assignment statement
STACKBASE	A WORD variable that represents the hardware stack base register (SS) of the iAPX 86/88
STACKPTR	A WORD variable that represents the hardware stack pointer register (SP) of the iAPX 86/88

iAPX 86/88 Built-In Functions

Identifier	Parameter List	Returned Value	Operation
BLOCK$INPUT	(*port,dest,count*)	none	Reads BYTE string from specified input port
BLOCK$INWORD	(*port,dest,count*)	none	Reads WORD string from specified input port
BLOCK$OUTPUT	(*port,source,count*)	none	Writes BYTE string to specified output port
BLOCK$OUTWORD	(*port,source,count*)	none	Writes WORD string to specified output port
CARRY	none	BYTE	Returns value of hardware CARRY flag
DEC	(*byte-expr*)	BYTE	Performs decimal adjust operation on BYTE value
PARITY	none	BYTE	Returns value of hardware PARITY flag
SCL	(*pattern,count*)	BYTE or WORD	Rotates an 8- or 16-bit pattern to the left with the CARRY flag
SCR	(*pattern,count*)	BYTE or WORD	Rotates an 8- or 16-bit pattern to the right with the CARRY flag
SIGN	none	BYTE	Returns value of hardware SIGN flag
ZERO	none	BYTE	Returns value of hardware ZERO flag

8087 Built-In Procedures and Functions

Identifier	Parameter List	Returned Value	Operation
GET$REAL$ERROR	none	BYTE	Returns contents of REAL error byte and clears it
INIT$REAL$MATH$UNIT	none	none	Initializes REAL math unit for subsequent operation
RESTORE$REAL$STATUS	(*save-area*)	none	Restores status of REAL math unit from the specified save area
SAVE$REAL$STATUS	(*save-area*)	none	Saves status of REAL math unit at the specified save area; reinitializes REAL error byte
SET$REAL$MODE	(*mode-word*)	none	Sets contents of REAL stack mode word

Compiler Controls

Control	Default	Action
CODE/NOCODE	NOCODE	Allow of prevent listing of approximate assembly code
COND/NOCOND	COND	List any text that is skipped
*DEBUG/NODEBUG	NODEBUG	Generate debug records in the object module
EJECT	paging is automatic	Force start of new page of printed output
IF/ELSEIF/ELSE/ENDIF	—	Establish actual conditional capability with conditions based on the value of switches

Compiler Controls (Cont'd.)

Control	Default	Action
INCLUDE(*file*)	not applicable	Include other source files as input to the compiler
*INTVECTOR/NOINTVECTOR	INTVECTOR	Create interrupt vector for each interrupt procedure in the module
INTERFACE	—	Allows PL/M to call procedures written in C, and vice versa
LEFTMARGIN (*n*)	LEFTMARGIN(1)	Set left margin of source input
LIST/NOLIST	LIST	Allow or prevent listing of source lines
*MOD86/MOD186	MOD86	Specifies which instruction set the object module will generate for the targeted processor
*OBJECT[(*file*)]/NOOBJECT	OBJECT(*source*.OBJ)	Specify a filename for the object module, or prevent the creation of an object module
*OPTIMIZE (*n*)	OPTIMIZE(1)	Control the level of optimization performed while generating object code
OVERFLOW/NOOVERFLOW	NOOVERFLOW	Specify whether overflow is to be detected in signed arithmetic
*PAGING/NOPAGING	PAGING	Format output onto pages with a heading that identifies the compiler and a page number
*PAGELENGTH (*n*)	PAGELENGTH(60)	Specify the maximum number of lines on a page
*PAGEWIDTH (*n*)	PAGEWIDTH(120)	Specify the maximum number of characters on a line
*PRINT[(*file*)]/NOPRINT	PRINT(*source*.LST)	Allow or prevent printed output, or select device or file to receive printed output
*RAM/ROM	RAM	Direct the object-module placement of all constants
SAVE/RESTORE	—	Save certain general controls on the stack before an INCLUDE control switches the input source to another file
SET/RESET (*switches*)	—	Set switch assignment
*SMALL/COMPACT/MEDIUM/LARGE	SMALL	Specify the segmentation scheme of a program
SUBTITLE('*subtitle*')	no subtitle	Put a subtitle on each page of printed output and cause a page eject
*SYMBOLS/NOSYMBOLS	NOSYMBOLS	List all identifiers in PL/M-86 source program and attributes
*TITLE('*title*')	module name in source	Put a title on each page of printed output
*TYPE/NOTYPE	TYPE	Include type records in object module
*XREF/NOXREF	NOXREF	Allow or prevent a cross-reference listing of source program identifiers